

Cling Updates

Vassil Vassilev, Princeton

<https://compiler-research.org>

Cling Overview

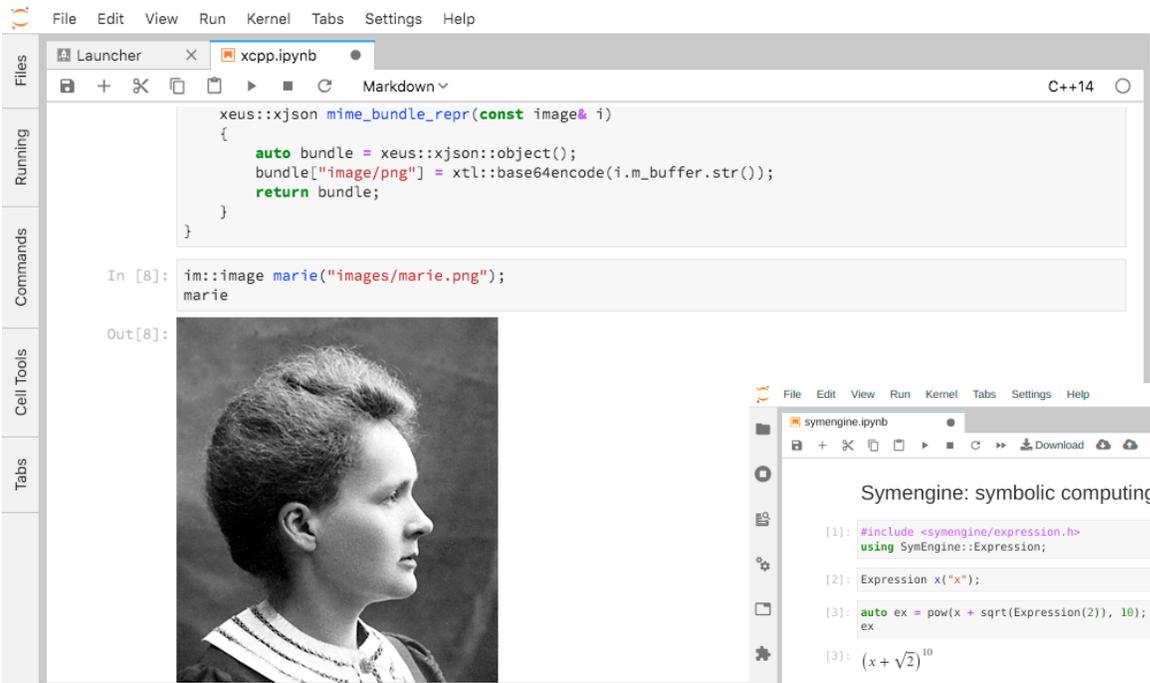
- The LLVM-based C++ interpreter used in ROOT6
- Enables interactivity, IO, PyROOT/cppyy
- Can be used as a standalone tool
- The C++ engine behind interactive C++ with Jupyter via zeus-cling

```
[root] ntuple->GetTitle()  
error: use of undeclared identifier 'ntuple'  
[root] TFile::Open("tutorials/hsimple.root"); ntuple->GetTitle()  
(const char *) "Demo ntuple"  
[root] gFile->ls();  
TFile**      tutorials/hsimple.root      Demo ROOT file with histograms  
TFile*       tutorials/hsimple.root      Demo ROOT file with histograms  
OBJ: TH1F    hpx      This is the px distribution : 0 at: 0x7fadbb84e390  
OBJ: TNtuple  ntuple   Demo ntuple : 0 at: 0x7fadbb93a890  
KEY: TH1F    hpx;1    This is the px distribution  
[...]  
KEY: TNtuple  ntuple;1  Demo ntuple  
[root] hpx->Draw()c
```

Four Cling Releases Since 2018

- [v0.6](#)
 - Enables Plugins
 - Adds automatic differentiations support with clad
 - Emulated thread local storage
- [v0.7](#)
 - Implement a mechanism allowing to redefine entities with the same name
 - CUDA support
 - Symbol resolver based on the binary object formats
- [v0.8](#)
 - Complete the C++ Modules used in ROOT's dictionaries
- [v0.9](#)
 - Upgrade to LLVM9
 - Improvements in the CUDA backend and
 - Improvements in C++ Module support
 - Reduced dependence on custom patches

Xeus-Cling. C++ in Notebooks



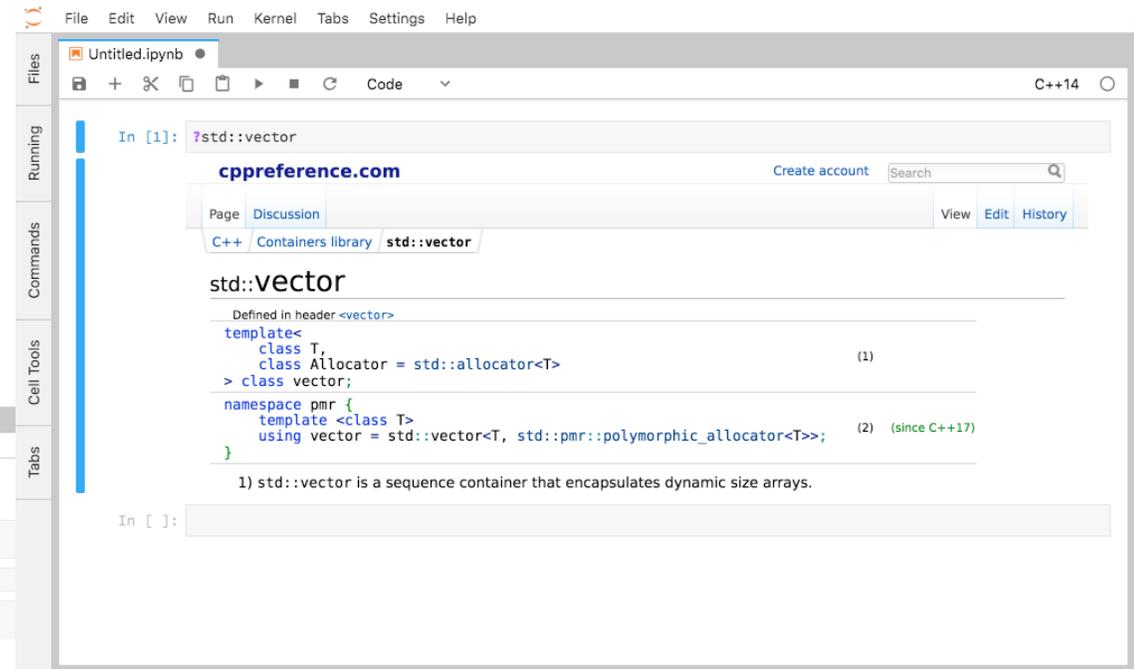
The screenshot shows a Jupyter notebook interface with a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The notebook has a tab labeled 'xcpp.ipynb'. The code cell contains C++ code using the Xeus library to load an image from a file. The output cell shows the command 'im::image marie("images/marie.png"); marie' and the resulting image of a woman's profile.

```
xeus::xjson mime_bundle_repr(const image& i)
{
    auto bundle = xeus::xjson::object();
    bundle["image/png"] = xtl::base64encode(i.m_buffer.str());
    return bundle;
}

In [8]: im::image marie("images/marie.png");
marie

Out[8]: 
```

Visualization of user-defined images



The screenshot shows a Jupyter notebook interface with a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The notebook has a tab labeled 'Untitled.ipynb'. The code cell contains the command '?std::vector', which has triggered a search for documentation. The output shows the cppreference.com website with search results for 'std::vector', including the class definition and a brief description.

```
In [1]: ?std::vector

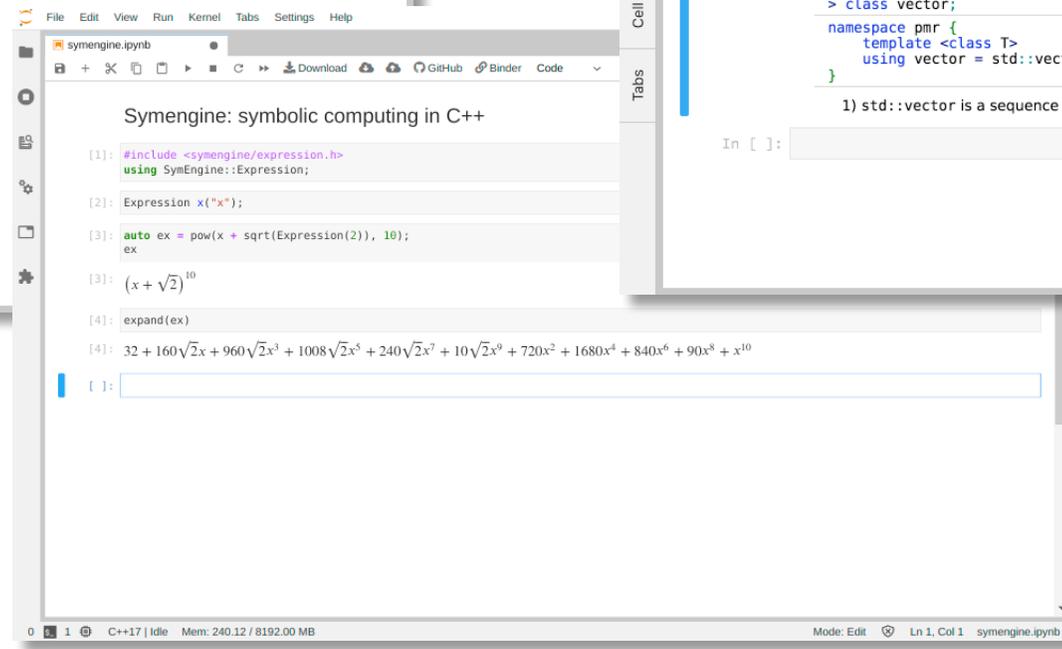
cppreference.com Create account Search

Page Discussion View Edit History
C++ Containers library std::vector

std::vector
Defined in header <vector>
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
namespace pmr {
    template <class T>
        using vector = std::vector<T, std::pmr::polymorphic_allocator<T>>;
}

1) std::vector is a sequence container that encapsulates dynamic size arrays.
```

Direct access to documentation



The screenshot shows a Jupyter notebook interface with a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The notebook has a tab labeled 'symengine.ipynb'. The code cell contains C++ code using the Symengine library for symbolic computation. The output shows the command 'expand(ex)' and the resulting expanded expression.

```
Symengine: symbolic computing in C++

[1]: #include <symengine/expression.h>
using SymEngine::Expression;

[2]: Expression x("x");

[3]: auto ex = pow(x + sqrt(Expression(2)), 10);
ex

[3]:  $(x + \sqrt{2})^{10}$ 

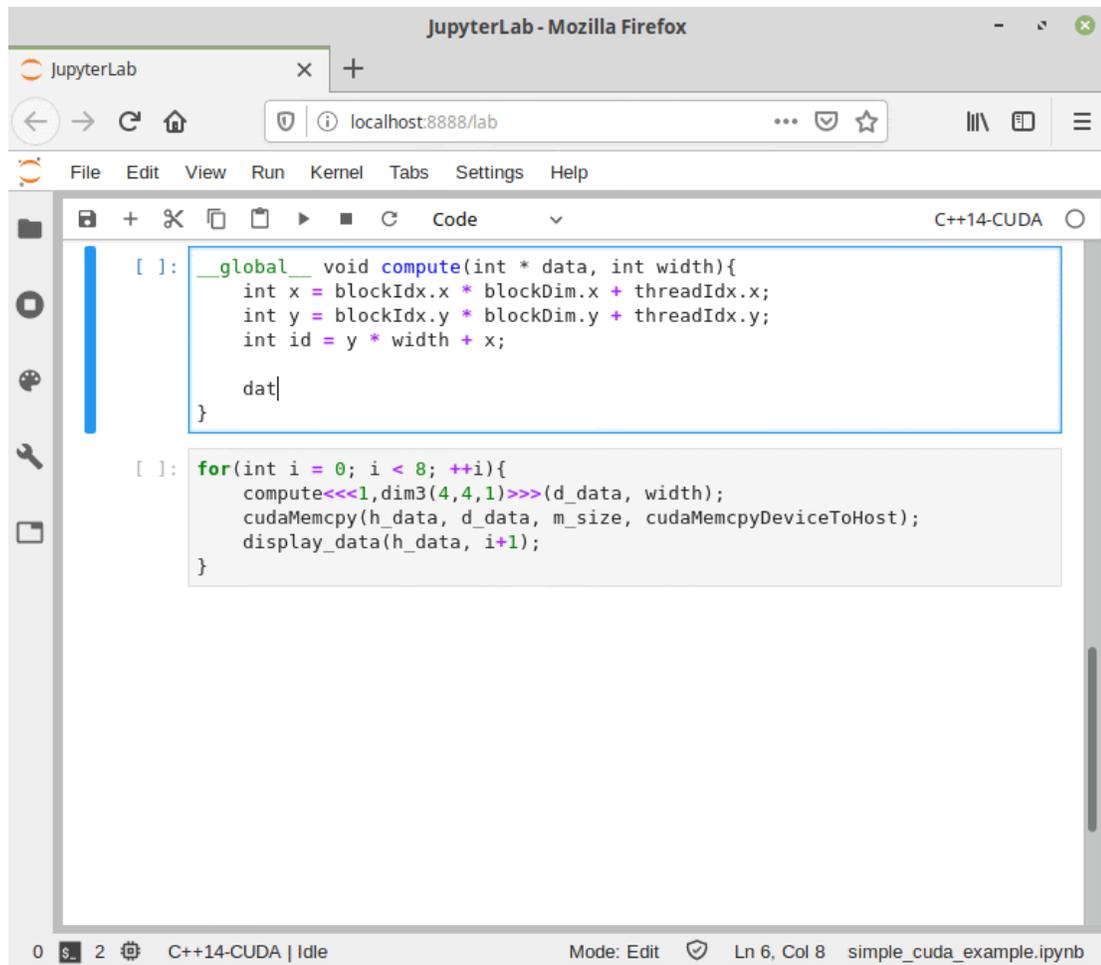
[4]: expand(ex)

[4]:  $32 + 160\sqrt{2}x + 960\sqrt{2}x^3 + 1008\sqrt{2}x^5 + 240\sqrt{2}x^7 + 10\sqrt{2}x^9 + 720x^2 + 1680x^4 + 840x^6 + 90x^8 + x^{10}$ 
```

Rich mime type rendering in Jupyter

S. Corlay, Quantstack, [Deep dive into the Xeus-based Cling kernel for Jupyter](#), May 2021, compiler-research.org

Interactive CUDA C++



JupyterLab - Mozilla Firefox

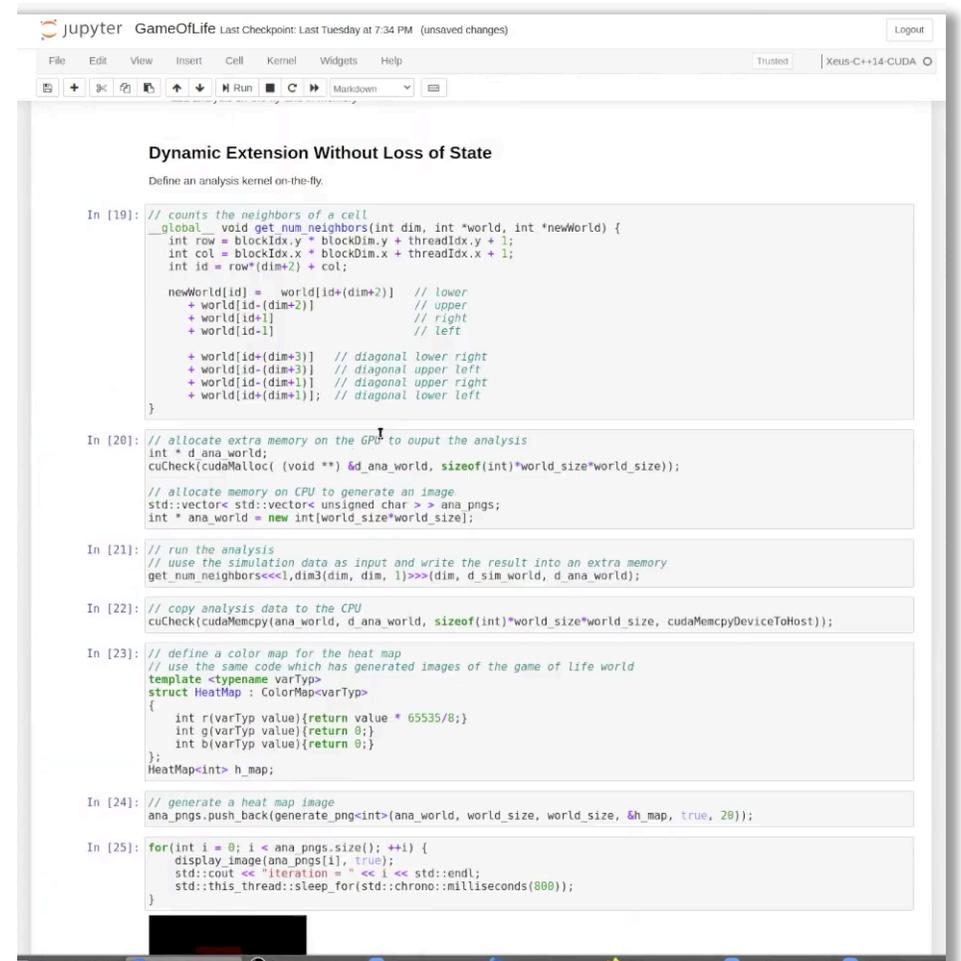
localhost:8888/lab

```
[ ]: global void compute(int * data, int width){
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    int id = y * width + x;

    dat
}

[ ]: for(int i = 0; i < 8; ++i){
    compute<<<1,dim3(4,4,1)>>>(d_data, width);
    cudaMemcpy(h_data, d_data, m_size, cudaMemcpyDeviceToHost);
    display_data(h_data, i+1);
}
```

0 2 C++14-CUDA | Idle Mode: Edit Ln 6, Col 8 simple_cuda_example.ipynb



jupyter GameOfLife Last Checkpoint: Last Tuesday at 7:34 PM (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Xeus-C++14-CUDA

Dynamic Extension Without Loss of State

Define an analysis kernel on-the-fly.

```
In [19]: // counts the neighbors of a cell
global void get_num_neighbors(int dim, int *world, int *newWorld) {
    int row = blockIdx.y * blockDim.y + threadIdx.y + 1;
    int col = blockIdx.x * blockDim.x + threadIdx.x + 1;
    int id = row*(dim+2) + col;

    newWorld[id] = world[id+(dim+2)] // lower
    + world[id-(dim+2)] // upper
    + world[id+1] // right
    + world[id-1] // left

    + world[id+(dim+3)] // diagonal lower right
    + world[id-(dim+3)] // diagonal upper left
    + world[id+(dim+1)] // diagonal upper right
    + world[id+(dim+1)]; // diagonal lower left
}

In [20]: // allocate extra memory on the GPU to output the analysis
int * d_ana_world;
cuCheck(cudaMalloc((void **) &d_ana_world, sizeof(int)*world_size*world_size));

// allocate memory on CPU to generate an image
std::vector<std::vector<unsigned char>> ana_pngs;
int * ana_world = new int[world_size*world_size];

In [21]: // run the analysis
// use the simulation data as input and write the result into an extra memory
get_num_neighbors<<<1,dim3(dim, dim, 1)>>>(dim, d_sim_world, d_ana_world);

In [22]: // copy analysis data to the CPU
cuCheck(cudaMemcpy(ana_world, d_ana_world, sizeof(int)*world_size*world_size, cudaMemcpyDeviceToHost));

In [23]: // define a color map for the heat map
// use the same code which has generated images of the game of life world
template <typename varTyp>
struct HeatMap : ColorMap<varTyp>
{
    int r(varTyp value){return value * 65535/8;}
    int g(varTyp value){return 0;}
    int b(varTyp value){return 0;}
};
HeatMap<int> h_map;

In [24]: // generate a heat map image
ana_pngs.push_back(generate_png<int>(ana_world, world_size, world_size, &h_map, true, 20));

In [25]: for(int i = 0; i < ana_pngs.size(); ++i) {
    display_image(ana_pngs[i], true);
    std::cout << "iteration = " << i << std::endl;
    std::this_thread::sleep_for(std::chrono::milliseconds(800));
}
```

S. Ehrig, HZDR, [Cling's CUDA Backend: Interactive GPU development with CUDA C++](https://arxiv.org/abs/2103.04441), Mar 2021, compiler-research.org

Automatic Language Bindings With Cling

cppyy: Yet another Python – C++ binder?!

- Yes, but it has its niche: *bindings are runtime*
 - Python is all runtime, so runtime is more natural
 - C++-side runtime-ness is provided by Cling
- Very complete feature-set (not just “C with classes”)
- Good performance on CPython; great with PyPy*

pip: <https://pypi.org/project/cppyy/>
conda: <https://anaconda.org/conda-forge/cppyy>
git: <https://github.com/wlav/cppyy>
docs: <https://cppyy.readthedocs.io/en/latest/>

For HEP users: *cppyy in ROOT is an old fork. It won't run all the examples here, doesn't work with PyPy, and has worse performance.*

(* PyPy support lags CPython

[1]

sil-cling: Interface with cppyy

- binds with cppyy through the direct inclusion of the latter's C header using dpp

```
1 //cling.dpp
2
3 #include "capi.h" // cppyy's C header
4
5 // D code ↓
6 import std.string : fromStringz, toStringz;
7
8 string resolveName(string cppItemName)
9 {
10     import core.stdc.stdlib : free;
11     // Calling cppyy_resolve_name ↓
12     char* chars = cppyy_resolve_name(cppItemName.toStringz);
13     string result = chars.fromStringz.idup;
14     free (chars);
15     return result;
16 }
```

```
→ ~ dub run dpp -- cling.dpp --keep-d-files -c
```

[2]

[1] W. Lavrijsen, LBL, [cppyy](https://compiler-research.org), Sep 2021, compiler-research.org

[2] A. Militaru, Symmetry Investments, [Calling C++ libraries from a D-written DSL: A cling/cppyy-based approach](https://compiler-research.org), Feb 2021, compiler-research.org

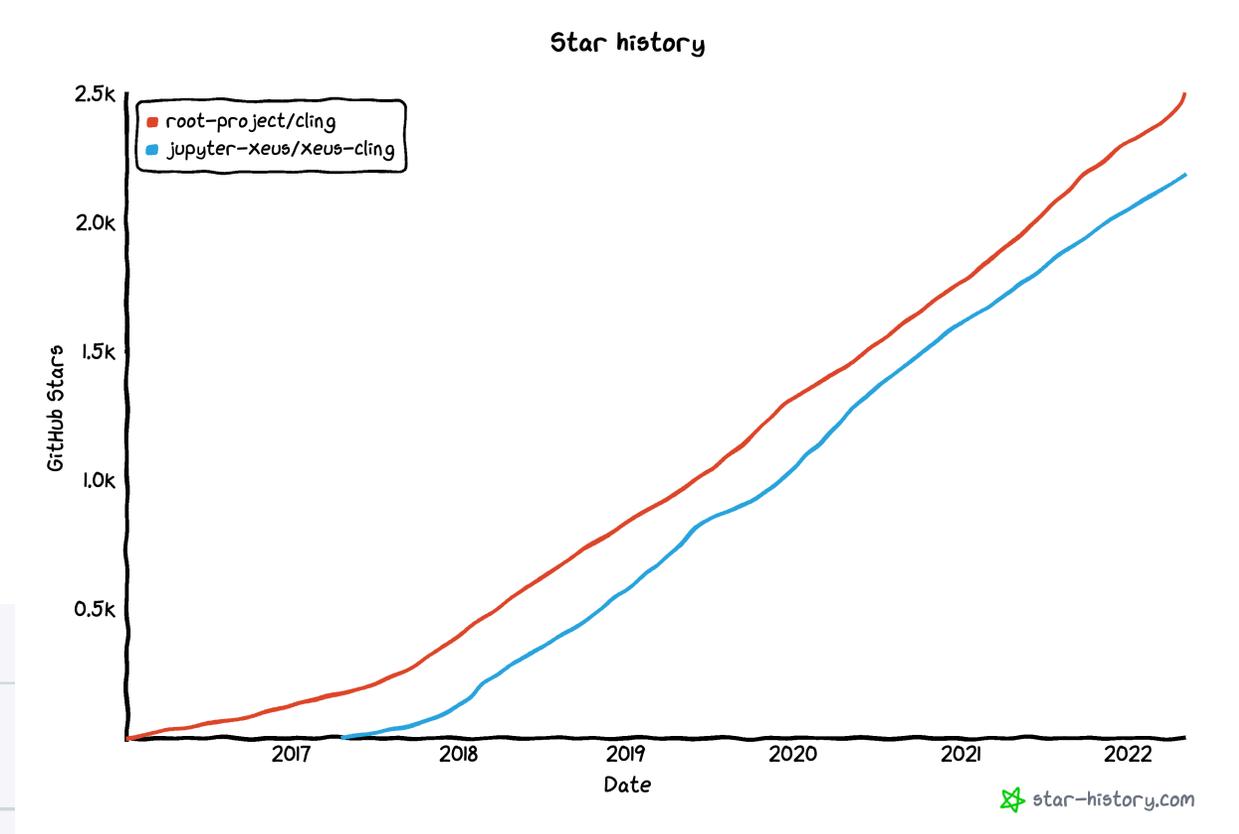
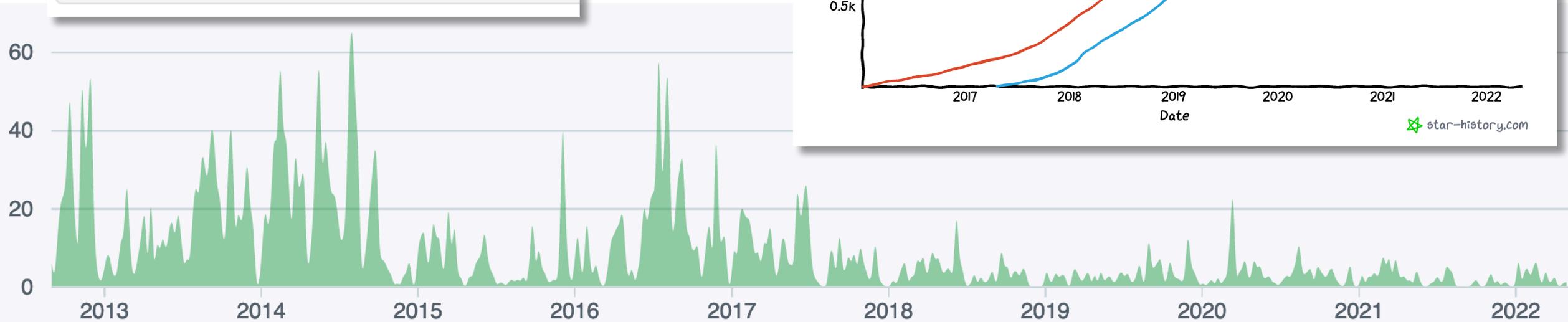
Repository Statistics

root-project/cling

ROOT-PROJECT/CLING

issue resolution 60 d open issues 49%

View the details



Moving Cling Closer to LLVM Orbit

- We have demonstrated the power of incremental compilation in ROOT on EB of data
- The growing community around Cling and outside of HEP has shown its relevance to the data science community in general
- Cling's growing community needs better integration in LLVM in terms of release cycles and a stronger connection between the two highly knowledgeable system software engineering communities – the one around LLVM, and the one around data analysis in HEP

The current work is partially supported by National Science Foundation under Grant OAC-1931408

Moving Cling Closer to LLVM Orbit. Clang-Repl

[2018] – We started thinking of dedicated resources to make Cling available to a broader audience

[2019] – We received an NSF award supporting this goal

[2020] – We laid our arguments in a “request for comment” document on the LLVM mailing [lists](#)

[2021] – Initial, minimally functional Cling called [clang-repl](#) landed in the LLVM repository and was released with LLVM13

[2022] – The LLVM13 upgrade in ROOT uses our work and reduces the codebase of Cling

Thank You!

Selected References

<https://blog.llvm.org/posts/2020-11-30-interactive-cpp-with-cling/>

<https://blog.llvm.org/posts/2020-12-21-interactive-cpp-for-data-science/>

<https://blog.llvm.org/posts/2021-03-25-cling-beyond-just-interpreting-cpp/>

<https://Compiler-Research.org>